Python tutorial cont. and Improving Networks

SETH HUANG

PYTHON TUTORIALS PART 4 AND IMPROVING YOUR NEURAL NETWORK

DOING STUFF TO A LIST: A WARM UP

```
1  ten_things = "Apples Oranges Crows Telephone Light Sugar"
2
3  print "Wait there are not 10 things in that list. Let's fix that."
4
5  stuff = ten_things.split(' ')
6  more_stuff = ["Day", "Night", "Song", "Frisbee", "Corn", "Banana", "Girl", "Boy"]
7
8  while len(stuff) != 10:
9      next_one = more_stuff.pop()
10     print "Adding: ", next_one
11     stuff.append(next_one)
12     print "There are %d items now." % len(stuff)
13
14 print "There we go: ", stuff
15
16 print "Let's do some things with stuff."
17
18 print stuff[1]
19 print stuff[-1] # whoa! fancy
20 print stuff.pop()
21 print ' '.join(stuff) # what? cool!
22 print '#'.join(stuff[3:5]) # super stellar!
```

DICTIONARY OH DICTIONARY: (IF YOU DON'T LIKE DICTIONARIES, THINK HASHTAG) TRY IT!

```
>>> things = ['a', 'b', 'c', 'd']
>>> print things[1]
b
>>> things[1] = 'z'
>>> print things[1]
z
>>> things
['a', 'z', 'c', 'd']
```

```
>>> stuff = {'name': 'Zed', 'age': 39, 'height': 6 * 12 + 2}
>>> print stuff['name']
Zed
>>> print stuff['age']
39
>>> print stuff['height']
74
>>> stuff['city'] = "San Francisco"
>>> print stuff['city']
San Francisco
```

MODULES, CLASSES AND OBJECTS

Module: A Python file with some functions or variables in it ..
1. You import that file.
2. And you can access the functions or variables in that module with the . (dot) operator.
Modules are a bit **like dictionaries in that you use "dot" to access the codes.**

Class: A class takes a group of functions **AND DATA** so you can access with "dot!"

YOU KNOW WHAT A MODULE IS, SO NOW LET'S SEE A CLASS EXAMPLE

```
1  class MyStuff(object):
2
3      def __init__(self):
4          self.tangerine = "And now a thousand years between"
5
6      def apple(self):
7          print "I AM CLASSY APPLES!"
```

That looks complicated compared to modules, but you should be able to make out how this is like a "mini-module" with "**MyStuff" having an apple() function in it**.

Make sure you have **__init__()** function and **self.tangerine** for setting the tangerine instance variable.

OBJECTS ARE WHAT YOU "INSTANTIATE" OUT OF CLASSES

It is similar to you "importing" a module. But now we do something similar for classes.

```
1  thing = MyStuff()
2  thing.apple()
3  print thing.tangerine
```

1. Python gets "MyStuff" and that is a "class" that you create.
2. You have the magical "__init__" function so Python knows you are "initializing" an object
3. The "self" is just the name you give it

CLASS EXAMPLE FOR YOU

```
1  class Song(object):
2
3      def __init__(self, lyrics):
4          self.lyrics = lyrics
5
6      def sing_me_a_song(self):
7          for line in self.lyrics:
8              print line
9
10 happy_bday = Song(["Happy birthday to you",
11                    "I don't want to get sued",
12                    "So I'll stop right there"])
13
14 bulls_on_parade = Song(["They rally around tha family",
15                         "With pockets full of shells"])
16
17 happy_bday.sing_me_a_song()
18
19 bulls_on_parade.sing_me_a_song()
```

IMPROVING NETWORKS AND WHY IT IS IMPORTANT

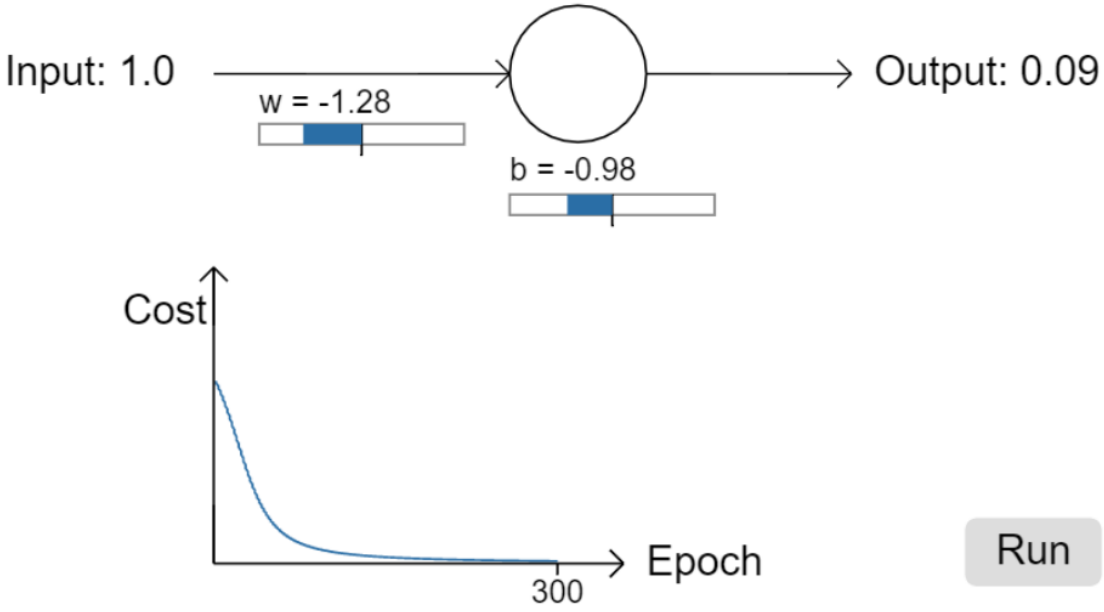
THERE ARE SEVERAL KEY ITEMS:
1. THE CROSS-ENTROPY COST FUNCTION
- PREVENTING TRAINING SLOW-DOWN

2. REGULARIZATION METHODS
- PREVENTING OVERFITTING THE TRAINING DATA

3. INITIALIZING THE WEIGHTS/ BIASES
- PICKING BETTER "STARTING POINTS"

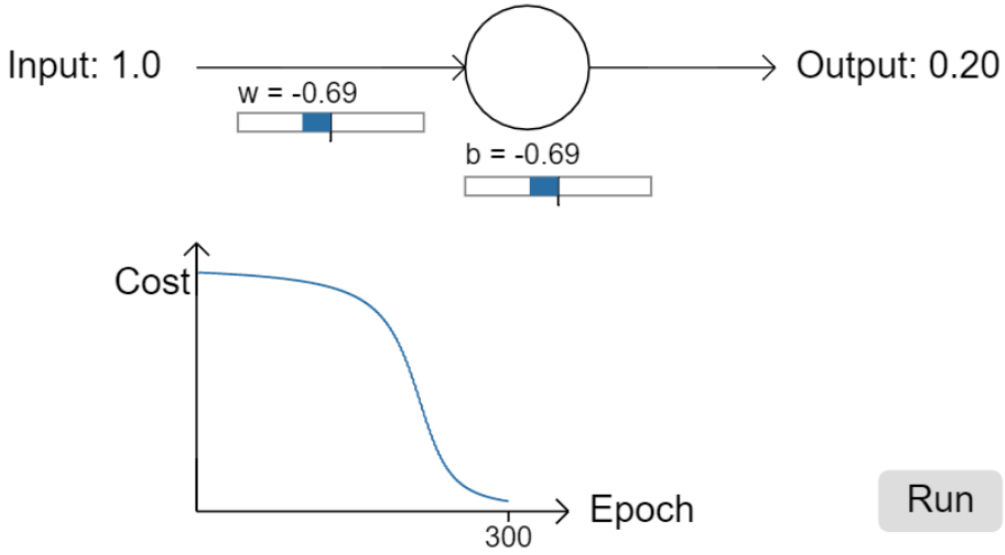
LET'S START WITH CROSS-ENTROPY, A HARD-SOUNDING BUT SIMPLE CONCEPT

The following is a simple neuron learning to get the desired output to be 0 with starting w = 0.6 and b = 0.9 and the initial output to be 0.86 and eta = 0.15

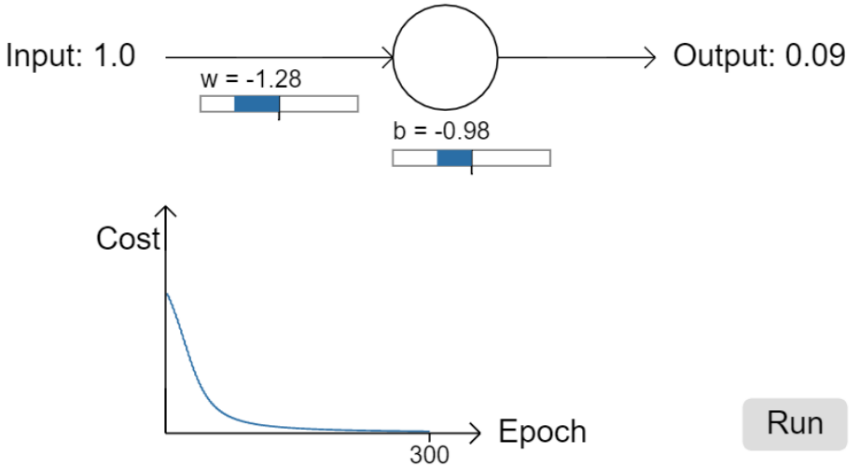


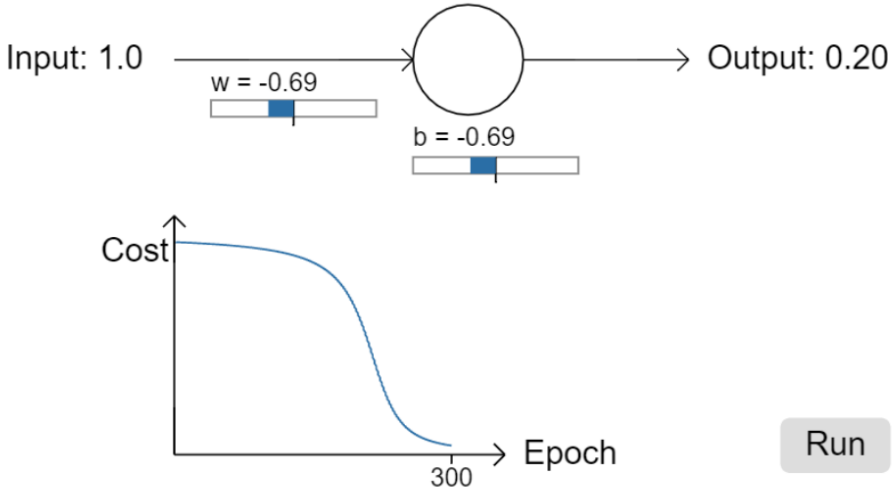
LET'S START WITH CROSS-ENTROPY, A HARD-SOUNDING BUT SIMPLE CONCEPT

The following is a simple neuron learning to get the desired output to be 0 with starting w = 2.0 and b = 2.0 and the initial output to be 0.98 and eta = 0.15



WHICH ONE IS BETTER AND FASTER?





SO, THE INITIAL WEIGHTS ARE BADLY WRONG, AND IT'S ACCOMPANIED WITH SLOW LEARNING RATE

Thought question: Is it similar to humans?

WHY IS LEARNING SLOW? WHAT DOES "SLOW MEAN" ON THE LANDSCAPE?

$$\partial C/\partial w \text{ and } \partial C/\partial b.$$

It means that the partial derivatives are small. The rates of change for cost with respect to weights and biases are small.

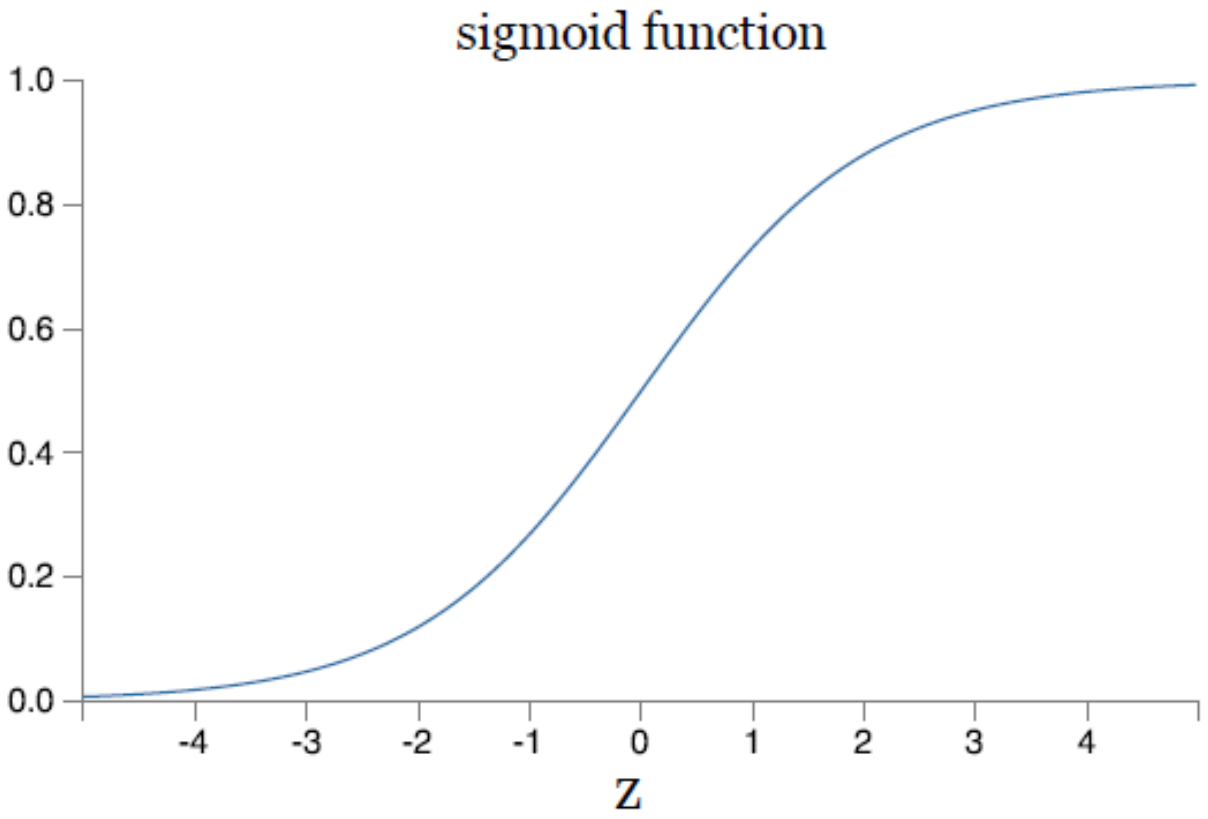
Recall the quadratic cost function:

$$C = \frac{(y-a)^2}{2},$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

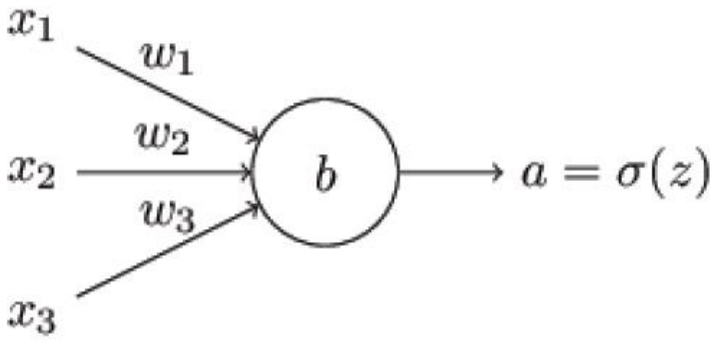
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

NOW, RECALL THE SIGMOID FUNCTION. WHAT DO YOU NOTICE?



THIS IS WHEN "CROSS-ENTROPY" RIDES IN LIKE A KNIGHT IN SHINY ARMOR

Keep in mind of the neuron here.



Cross-Entropy Cost Function:

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)],$$

"CROSS-ENTROPY" COST FUNCTION… UM…HOW IS IT A "COST FUNCTION?"

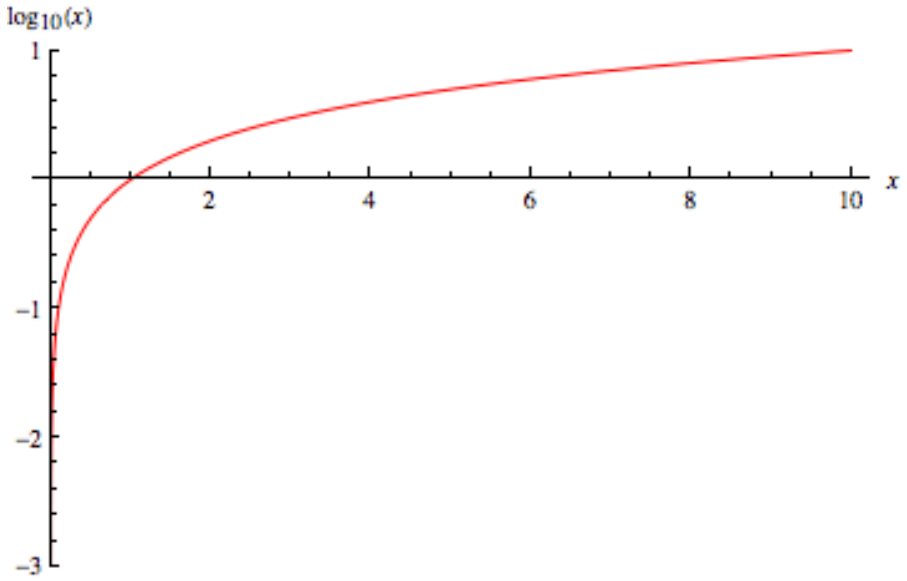
$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)],$$

What is a cost function?

- Non-negative
- Increasing with respect to errors in each sample

"CROSS-ENTROPY" COST FUNCTION… UM…HOW IS IT A "COST FUNCTION?"

$$C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right],$$



We also know that a (the output) has to be [0, 1]. WHY?

NOW, IMAGINE THAT Y = 0 AND OUTPUT ~= 0. THAT IS, THE MODEL IS DOING A GOOD JOB.

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)],$$

What happens?

Summing up, the cross-entropy is positive, and tends toward zero as the neuron gets better at computing the desired output!

NOW LET'S DEAL WITH THE "SLOW-DOWN" PROBLEM

$$C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right],$$

$$a = \sigma(z)$$

Let's see the partial derivatives of Cost with respect to weights:

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j} \\ &= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j.\end{aligned}$$

ORGANIZE IT A BIT MORE:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x \frac{\sigma'(z) x_j}{\sigma(z)(1-\sigma(z))} (\sigma(z) - y).$$

WITH: $\sigma(z) = 1/(1+e^{-z}),$

THEN: $\sigma'(z) = \sigma(z)(1-\sigma(z)).$

FINALLY:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$

COMPARED TO:

CROSS-ENTROPY:

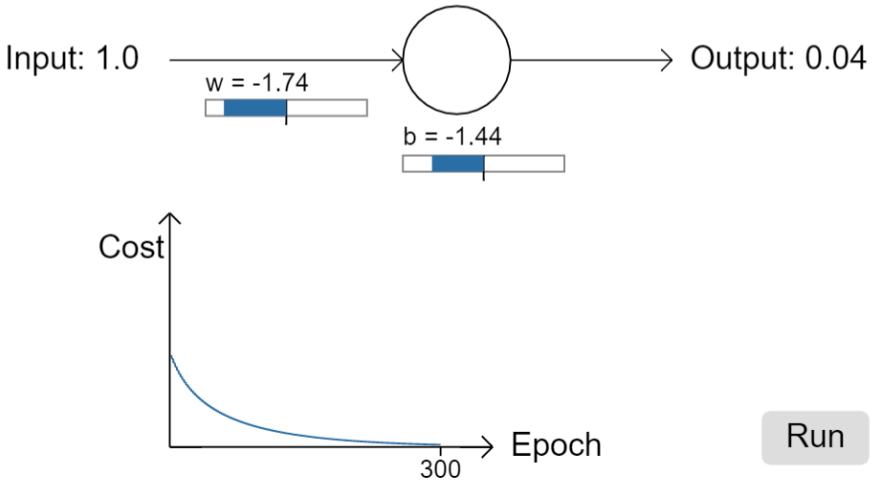
$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$

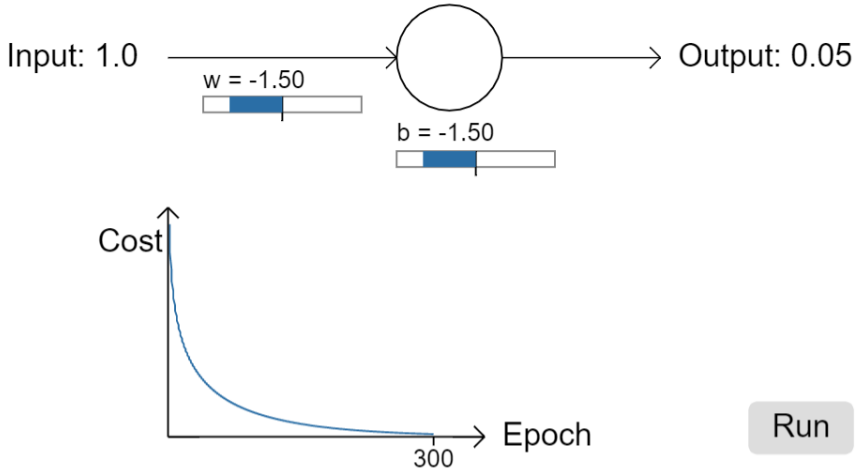
VANILLA:

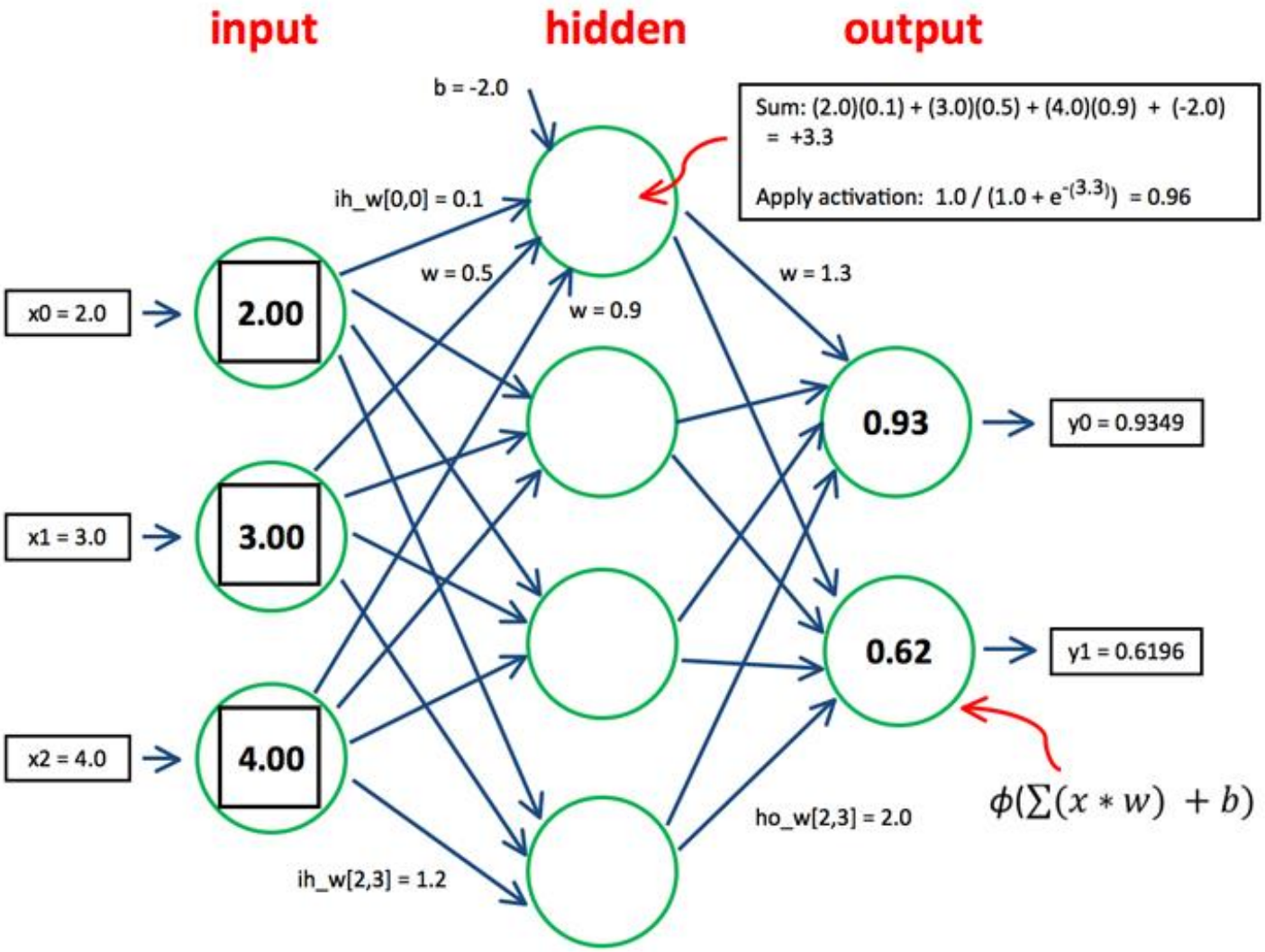
$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z) = a\sigma'(z),$$

WHAT HAPPENED TO THE SLOPES? IT IS NOW CONTROLLED BY THE ERRORS OF THE OUTPUT!

WHICH ONE IS BETTER AND FASTER? (THE SECOND ROUND!)






input
hidden
output
b = -2.0
Sum: (2.0)(0.1) + (3.0)(0.5) + (4.0)(0.9) + (-2.0)
= +3.3
Apply activation: 1.0 / (1.0 + e^-(3.3)) = 0.96
ih_w[0,0] = 0.1
w = 0.5
w = 0.9
w = 1.3
x0 = 2.0
2.00
x1 = 3.0
3.00
x2 = 4.0
4.00
0.93
0.62
y0 = 0.9349
y1 = 0.6196
ho_w[2,3] = 2.0
ih_w[2,3] = 1.2
$\phi(\sum(x * w) + b)$

NOW LET'S SEE IF YOU CAN USE CROSS-ENTROPY FOR MNIST DIGITS. WHAT IS THE ACCURACY?

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.5, evaluation_data=test_data,
... monitor_evaluation_accuracy=True)
```

NOW, CHANGE THE NUMBER OF NEURONS TO 100. WHAT DO YOU GET?

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.5, evaluation_data=test_data,
... monitor_evaluation_accuracy=True)
```

DOES CROSS-ENTROPY ACTUALLY IMPROVE THE ACCURACY?

=> What is the problem we are tackling here?

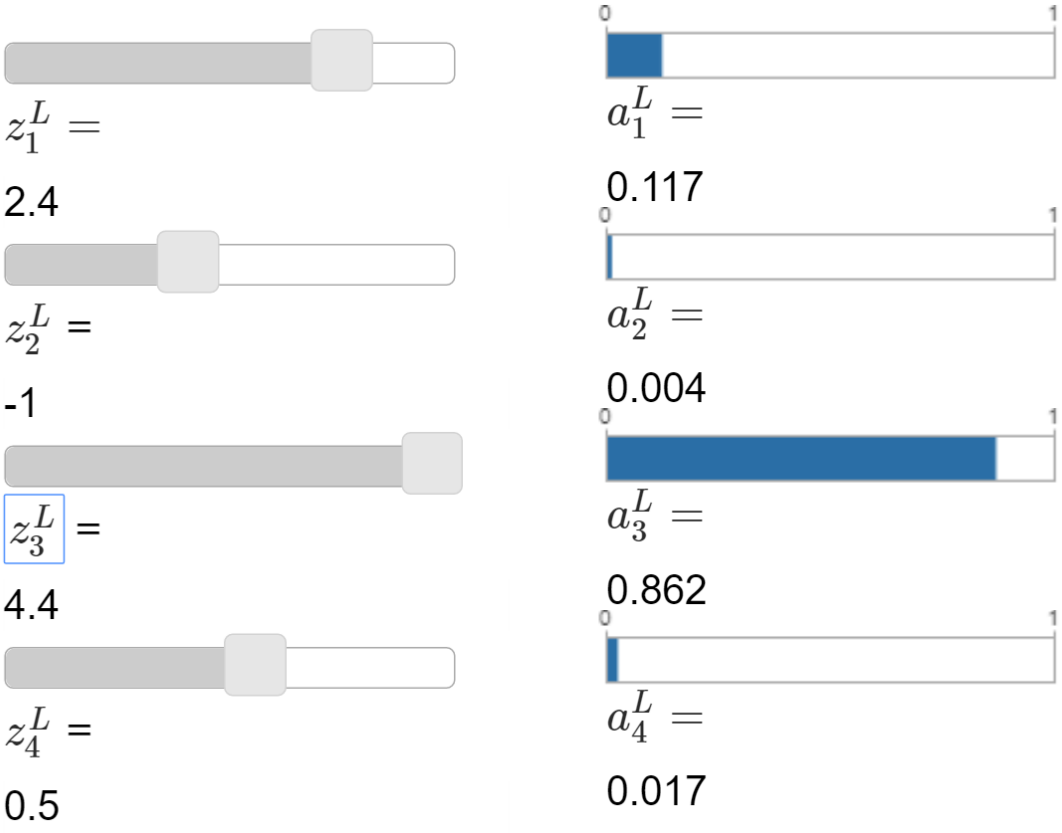
BEFORE WE GO FURTHER, LET'S LOOK AT A VERY POPULAR FINAL LAYER: SOFTMAX

Remember our good buddy SIGMOID?
We are using something different.

$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L.$$

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

WHAT DO YOU NOTICE?



WHAT DO YOU NOTICE ABOUT SOFTMAX?

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

1. The outputs add up to 1
2. Can be thought of as a probability distribution
3. The cost derivative using Softmax also reduces the learning slow-down problem, but we will not elaborate on that for now.

WHAT DO USE?

1. Sigmoid is a traditional activation function and has worked well
2. For deep learning, you can use cross-entropy and softmax approach, which we will do in this semester.

LASTLY, OVERFITTING AND REGULARIZATION INTRO

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data[:1000], 400, 10, 0.5, evaluation_data=test_data,
... monitor_evaluation_accuracy=True, monitor_training_cost=True)
```

This code is monitoring different aspects of the model training.

COST ON TRAINING DATA: NO PROBLEM HERE

ACCURACY ON THE TEST DATA: HOW ABOUT IT?